CHAIR OF DECENTRALIZED INFORMATION SYSTEMS & DATA MANAGEMENT

TECHNICAL UNIVERSITY OF MUNICH

Thesis proposal

Concurrent Range-Locking

Author: **Thua-Duc Nguyen**

Supervisor: **Prof. Dr. Viktor Leis**

Advisor: **Lam-Duy Nguyen**

1 Introduction

In many popular systems, such as file or database systems, the ability to acquire exclusive locks on successive values is essential. Range locks provide this capability. They play an important role in file systems, operating systems, and databases [1, 2, 3]. Unlike traditional single-lock methods, range locks offer a more refined approach to resource access management. By dividing a shared resource into smaller segments, range locks allow multiple writers to modify different segments simultaneously. This approach overcomes the limitations and bottlenecks of single-lock approaches and enhances concurrency performance.

2 Motivation

Recently, the Linux kernel community has been considering using range-locking techniques to replace the `mmap_lock` [1, 4, 5]. The `mmap_lock` uses a per-process semaphore to control access to the whole `mm_struct` [6] and serialize changes to address spaces. Despite previous efforts to overcome the scalability issues of `mmap_lock`, a resolution is yet to be found [5].

In the context of database management systems, range locks also offer a solution to the issue of coarse-grained locking in large databases and indexes. With the database size increasing exponentially, there are better options than locking the entire index, considering that such a coarse-grained locking mechanism inherently blocks other transactions from progressing, leading to poor throughput and high latency. By focusing on specific segments within these structures, range locks enable multiple transactions to proceed concurrently on different segments, effectively reducing memory overhead and lock acquisition bottlenecks [2].

3 Related Work

Previous research has explored various approaches to range lock [7, 8, 9]. The current implementation of range lock in the Linux kernel uses a range tree that keeps track of acquired ranges and an internal spin lock to protect it [7]. Since every range request relies on this single spinlock, it becomes a point of contention.

Song et al. [8] try to improve the Linux kernel's implementation by combining a skip list with a spinlock to manage locked ranges. This technique leverages the skip list, which is more lightweight and efficient than the interval tree and can still conduct intensive searches for overlapping ranges. Despite these advancements, the problem of contention points still requires resolution.

In another research, Kogan et al. [9] designed a range lock based on a concurrent linked list, where each node represents an acquired range. This design achieves a lock-free mechanism for range locks, effectively addressing the pitfalls of existing range locks. However, insertion and lookup operations on the linked list are less efficient than tree-like structures.

4 Approach

In this research's scope, we propose a new concurrent range-locking design that leverages a probabilistic concurrent skip list [10, 11]. It consists of two main functions:

- **try_lock**: The `try_lock` function searches for the required range (`[start, start+len]`) in the skip list. If an overlapping range exists, indicating another thread is modifying that range, the requesting thread must wait and retry. If not, the range is added to the list, signaling that the range is reserved.
- **release_lock**: The `release_lock` function releases the lock by finding the address range in the skip list and removing it accordingly.

Our range lock design also utilizes the per-node lock instead of an interval lock, thus addressing the bottleneck problem of the spinlock-based range lock and maintaining the lock's high level of performance.

5 Evaluation

The proposed approach will be evaluated under these evaluation criteria:

- **Performance:** We will test the range lock mechanism under increasing load and concurrent accesses to measure its performance.
- **Correctness:** We will ensure the consistency and correctness of data accesses, especially when there are overlapping data ranges and concurrent operations.
- **Comparison:** We will compare the performance of the proposed solution with existing state-of-the-art approaches.

6 Expected Outcome

We aim to develop a concurrent range-locking mechanism that performs better than the existing range locks. The evaluation results will provide insights into the performance characteristics and potential trade-offs of the proposed mechanism.

7 Resources

We will need 32 cores and 32 GB of RAM for one month. These resources allow us to perform thorough tests under heavy contention and multithreaded scenarios.

Bibliography

[1] J. Corbet. "Range reader/writer locks for the kernel". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/724502/.

[2] G. Graefe. "Hierarchical locking in B-tree indexes". In: *On Transactional Concurrency Control*. Springer, 2007, pp. 45–73.

[3] C.-G. Lee, S. Noh, H. Kang, S. Hwang, and Y. Kim. "Concurrent file metadata structure using readers-writer lock". In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, pp. 1172–1181.

[4] M. Rybczynska. "Introducing maple trees". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/845507/.

[5] J. Corbet. "The ongoing search for mmap_lock scalability". In: *LWN.net* (2022). Accessed: 2024-04-21. URL: https://lwn.net/Articles/893906/.

[6] S. Boutnaru. "Linux Kernel — mm_struct". In: *medium.com* (2023). Accessed: 2024-04-21. URL: https://medium.com/@boutnaru/linux-kernel-mm-struct-fafe50b57837.

[7] J. Kara. "Implement range locks". In: *lkml.org* (2013). Accessed: 2024-04-21. URL: https://lkml.org/lkml/2013.

[8] X. Song, J. Shi, R. Liu, J. Yang, and H. Chen. "Parallelizing live migration of virtual machines". In: *Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 2013, pp. 85–96.

[9] A. Kogan, D. Dice, and S. Issa. "Scalable range locks for scalable address spaces and beyond". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. 2020, pp. 1–15.

[10] H. Maurice, L. Yossi, L. Victor, and S. Nir. "A provably correct scalable concurrent skip list". In: *Conference On Principles of Distributed Systems (OPODIS). Citeseer*. Vol. 103. 2006.

[11] H. Maurice, S. Nir, L. Victor, and S. Michael. *The art of multiprocessor programming*. Newnes, 2020.